Chapter 6
Load Balancing for Multi-cloud

Gabriel Iuhasz, Pooyan Jamshidi, Weikun Wang and Giuliano Casale

6.1 Introduction

Load balancing is an integral part of software systems that require to serve requests with multiple concurrent computing resources such as servers, clusters, network links, central processing units or disk drives. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource. It can also lead to a higher reliability through redundant resources. Load balancing typically involves two major components: (i) a controller, a piece of software or hardware controlling the routing of requests to the backend resources according to an specific routing policy; (ii) a reasoner that determines the routing policy. The policy can be set at design-time based on the result of the reasoner or at runtime based on periodic observation of response time and throughput.

The MODAClouds Load Balancer (Fig. 6.1) is a component for dispatching requests from end users to application servers following certain load balancing policies. It consists of a load balancing controller and a reasoner. The controller extends

G. Iuhasz (✉)
Institute E-Austria Timisoara and West University of TimiŞoara,
B-dul Vasile Pârvan 4, 300223 TimiŞoara, Romania
e-mail: iuhasz.gabriel@info.uvt.ro

P. Jamshidi · W. Wang · G. Casale
Department of Computing, Imperial College London,
180 Queens Gate, SW7 2AZ London, UK
e-mail: p.jamshidi@imperial.ac.uk

W. Wang
e-mail: weikun.wang11@imperial.ac.uk

G. Casale
e-mail: g.casale@imperial.ac.uk

E. Di Nitto et al. (eds.), *Model-Driven Development and Operation of Multi-Cloud Applications*, PoliMI SpringerBriefs,
DOI 10.1007/978-3-319-46031-4_6

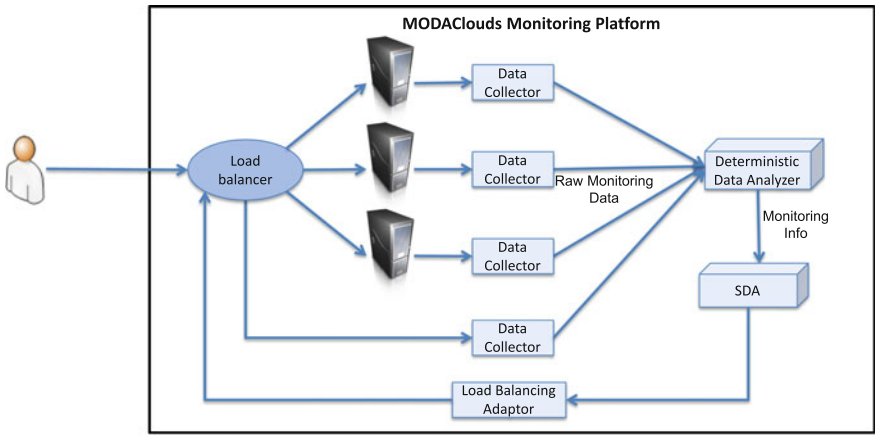

Fig. 6.1 The MODAClouds Load Balancer

well known open source load balancing and proxying for TCP and HTTP-based applications called HAProxy.[1]

6.2 Load Balancing Controller

In MODAClouds, we developed pyHrapi,[2] a set of REST APIs to interact with the core HAProxy engine. pyHrapi essentially controls the behavior of HAProxy through easy to use APIs easing the way self-adapting components of MODAClouds needs to interact with load balancing component either for configuration update or controlling its behavior at runtime.

6.3 Load Balancing Reasoner

MODAClouds Load Balancer uses the Weighted Round Robin policy, which dispatches requests to each server proportionally based on the assigned weights and in circular order. At runtime, algorithms proposed in [1] are implemented [2] to change the weights of the servers in order to optimize the revenue of the system. The revenue is defined as weighted throughput of different classes of users. The support of multi-class reasoning is useful in real applications when the users have different privileges, e.g. golden, silver and bronze, which stand for different levels of services. Such levels of service can be formalised by means of SLAs. In the reasoner component, the

[1] http://www.haproxy.org/.

[2] https://github.com/ieat/MODAClouds-loadbalancer-controller.

calculation of throughput and response time is based on log data analysis of the load balancing controller. At runtime, the per request logs are accumulated in the load balancing controller log file based on the requests hitting backend resources.

Data collector In order to observe the change of the system, we have developed a log file collector for Haproxy. This log file collector continuously extracts information from the log based on a predefined regular expression. Metrics like response time, arrival and departure time-stamps, request type and session IDs are particularly useful for the load balancing analysis to examine the processing requirement of each type of request on different types of servers. This Haproxy log data collector is integrated into Tower 4Clouds (see Chap. 5) and sends data to the Deterministic Data Analyzer (DDA) component.

Demand estimation The logs collected from the Haproxy log data collector are sent to the Haproxy specific demand estimation SDA from the DDA for analysis. We have developed two Haproxy specific demand estimation SDAs to obtain the demands of different classes of users: Complete Information (CI) and Utilization-based Regression (UBR). The CI method requires both the arrival time-stamps and departure time-stamps of each requests. The UBR, on the other hand, needs CPU utilization on the application server as well as the throughput of the requests. The demands obtained from either SDA will be used by the Load balancing adaptor to obtain the optimal weights for each backend resources.

In particular, we obtain the demand of different classes of users by evaluating the requests they send during one session. Here, we assume users have a similar behaviour for sending the requests. We achieve this by grouping the requests by the session IDs and examine if there are common requests among different sessions.

6.4 Multi-cloud Load Balancing

Local Load Balancing (LLB), also called cluster-level load balancing or intra-Cloud load balancing (see previous section), provides load balancing between VMs, which are inside a Cloud service or a virtual network (VNet) within a regional zone. However, there are several motivations for multi-Cloud (inter-Cloud) load balancing:

- *Failover*: An organization intends to provide highly reliable services for its customers. This can be realized by figuring out backup services in case their primary service goes down. A common architectural pattern for service failover is to provide a set of identical interfaces and route service requests to a primary service endpoint, with a list of one or more replicated ones. If the primary service goes down for a reason, requesting clients are routed to the other Cloud.
- *Gradual enhancement/graceful degradation*: Allocate a percentage of traffic to route to a new interface, and gradually shift the traffic over time.
- *Application migration to another Cloud*: A profile can be setup with both primary and secondary interfaces, and a weight can be specified to route the requests to each interface.

- *Cloud bursting*: A Cloud service can be expanded into another private or public Cloud by putting it behind a multi-Cloud load balancer profile. Once there is a need for extra resources, it can be added (or dynamically removed) and specify what proportion of requests goes to newly provisioned resources.
- *Internet scale services*: In order to load balance endpoints that are located in different Clouds across the world, one can direct incoming traffic to the closest port in terms of the lowest latency, which typically corresponds to the shortest geographic distance.
- *Fault tolerance*. A fault tolerant Cloud application detect failed components and fail over to another Cloud until a failure is resolved. It not only depends on deployment strategies but also on application level design strategies, for example, a reliable application may degraded and partly route the request to another Cloud at the same time.

6.4.1 Usage Scenario of Multi-cloud Load Balancing

Once a failure in a Cloud occurs, traffic can be redirected to VMs running in another Cloud. Multi-Cloud load balancing can facilitate this task. It allows to automatically manage the failover of traffic to another Cloud in case the primary Cloud fails. When configuring multi-Cloud load balancing, we need to provide a new global load balancer in front of the local ones in each Cloud. Global load balancer abstracts load balancing one level up the local level. The global load balancer maps to all the deployments it manages. Within global load balancer, the weights for the load balancing policy determine the priority of the deployments that users will be routed to a deployment. The global load balancer monitors the endpoints of the deployments and notes when a deployment in specific Cloud fails. At failure, global load balancer reasoner will change the weights and route users to other Cloud (Fig. 6.2).

6.5 Load Balancing and Failure Management

A requirement for the runtime platform is to be robust in case of failures of individual components. In particular, a failure of the load balancer could cause a major loss of connectivity for an entire cluster of machines. To prevent this situation, it is needed to replicate the load balancer and initiate an automatic fail-over switch to the backup load balancer in case the main load-balancer fails.

In MODAClouds, we use Trigger technology to handle load balancer failure. Triggers are functions that allow an action in Cloud Orchestrator to initiate a second action. A trigger is written as a block of code that run either before an event occurs (a pre trigger) or after an event occurs (a post trigger). One advantage is that a pool of images can be created and automatically started in the event of a fault. This reduces the requirement for a System Administrator to be involved and as a result reduce

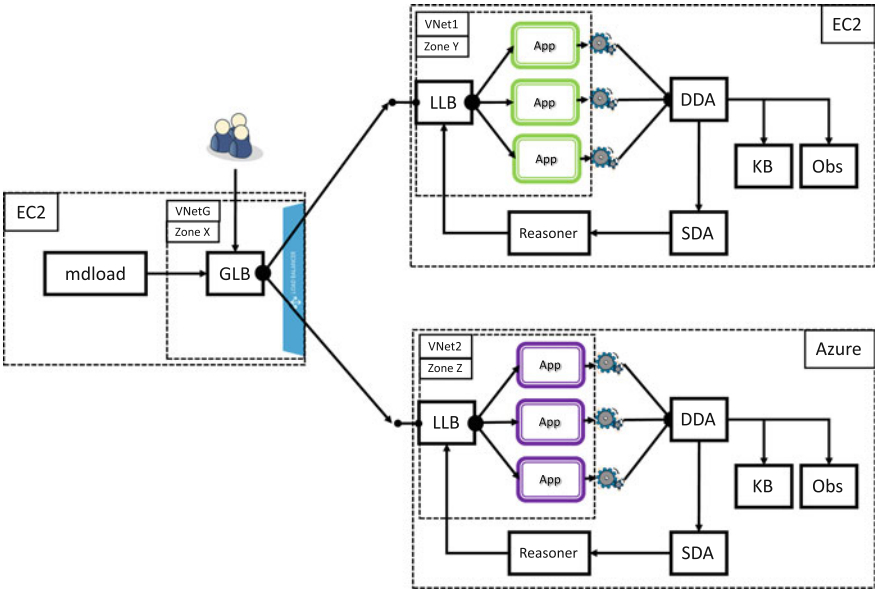

Fig. 6.2 Overview of multi-Cloud load balancing in MODAClouds

the overhead cost as well as providing a fast automatic response. Also as a snapshot of a disk is taken, if required a roll back in the event of changes or faults can be quick using this taken snapshot and the provided tools. We also use triggers to add the newly started VM into the load balancer, that would work in conjunction with the previously created triggers. In the experiments we performed we exploited the trigger implementation presented in Chap. 15.

6.6 Conclusion

The observations with our experimental study can be summarized as follows:

- *Flexible load balancing*. The approach enabled adaptive changes in the weights according to the heterogeneity of the resources in each Cloud.
- *Reduce application downtime*. The approach improved the availability of Cloud-based applications by automatically directing user access to a new location anytime there is a congestion in a Cloud.
- *Improved performance*. The approach made application more responsive by directing access to an application according to the weights

Further details on implementation, experimental results, and the interconnection with the other runtime components of MODAClouds can be found in [3].

References

1. Anselmi J, Casale G (2013) Heavy-traffic revenue maximization in parallel multiclass queues. Perform Eval
2. Wang W, Casale G (2014) Evaluating weighted round robin load balancing for cloud web services
3. Iuhasz G et al (2015) Runtime environment final release. Modaclouds Deliverable D6.5.3